Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

9-Bit Voltage Output DAC Datasheet DAC9 V 2.2

Copyright © 2001-2019 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52						
	0	0	2	151	0	1

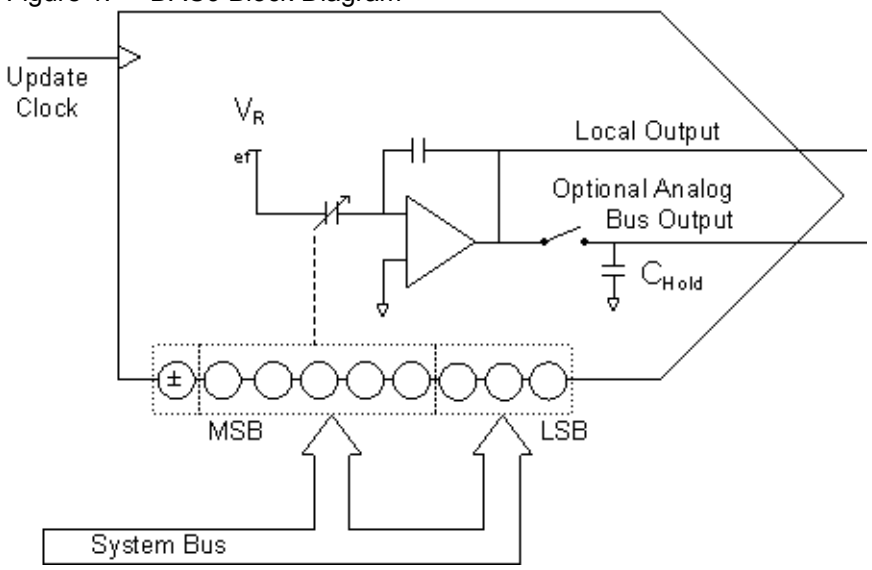
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- 9-bit resolution
- Voltage output
- 2’s complement, offset binary and sign/magnitude input data formats
- Sample and hold for analog bus and external outputs
- Update rates 54 ksps

The DAC9 User Module translates digital codes to output voltages. It translates digital codes to output voltages at an update rate of up to 54k samples per second. The Application Programming Interface (API) supports offset-binary, 2’s-complement, and register-image (sign-and-magnitude) data formats for maximum flexibility. Offset compensation is employed to minimize conversion error.

Figure 1. DAC9 Block Diagram

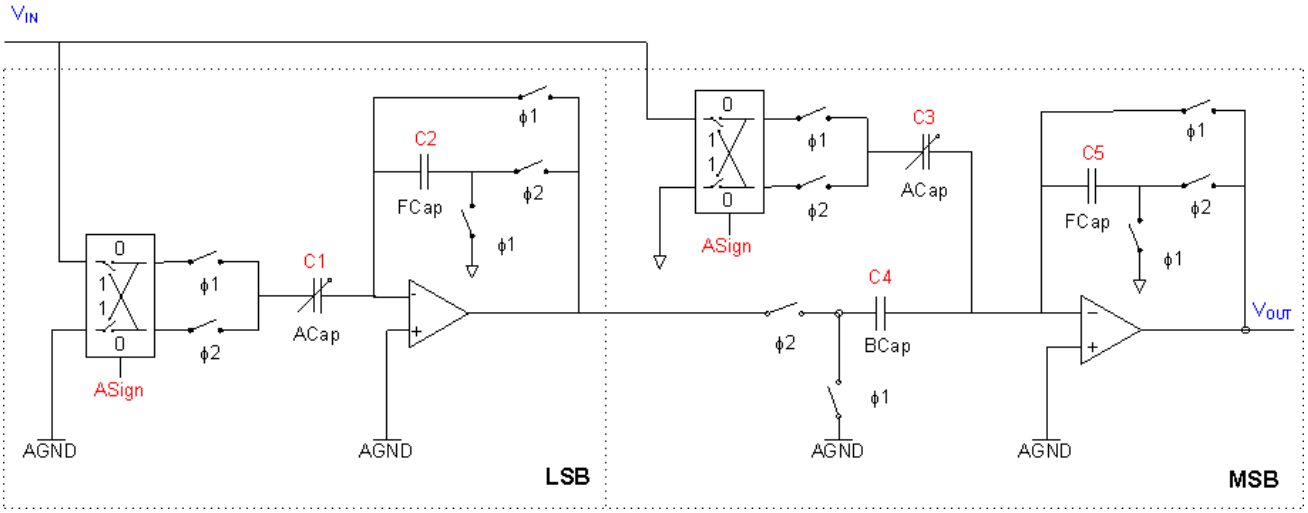


Functional Description

The DAC9 User Module converts digital codes into analog output voltages. The digital codes are represented as numbers in 2's complement form, ranging from -255 to +255. Alternatively, input codes may be represented in offset-binary form, as a number ranging from 0 to 510. Several output voltage ranges are possible, depending on the value selected for a system-level parameter, RefMux.

The DAC9 User Module maps onto any two analog PSoC blocks. These blocks are designated LSB and MSB. The LSB block, or stage, is coupled to the MSB stage through that block's "BCap" capacitor, C_4. Internally, the operation is based on sign-and-magnitude format. The five most significant magnitude bits set the value of C_3, an array of binary-weighted capacitors shown in the simplified schematic below. The two least significant magnitude bits set the value of C_1. C_3 assumes values from zero to 31 units and C_1 assumes values from the set {0, 4, 8, 12, 16, 20, 24, 28} in units of capacitance. The reference voltage, which may be inverted by the ASign bit, is scaled in each stage by the ratio of the magnitude capacitors, C_1 and C_3, to the feedback capacitors, C_2 and C_5, respectively. Each has a nominal capacitance of 32 units. The output of the LSB stage is further scaled by the ratio of the coupling capacitor, C_4, to the C_5 feedback capacitor.

Figure 2. Simplified Schematic of the DAC9



The hardware performs offset compensation in each update cycle. Switches controlled by Φ_1 and Φ_2 configure the opamps as a unity-gain followers during Φ_1. In this configuration, the offset voltages appear at the summing nodes, charging the various ACaps, BCaps, and FCaps. As reconfigured in Φ_2, the circuit inverts the offset charges on these capacitors, effectively canceling the offset voltages.

On every update cycle, V_{out} slews between the opamp offset voltage (during Φ_1) and the desired voltage (settled during Φ_2); a direct result of offset compensation. One way to mitigate this price for increased accuracy is by employing the sample-and-hold circuit associated with the output bus. V_{out} charges both the load and the hold capacitor (C_{Hold} in the DAC9 block diagram), during the last half of Φ_2. C_{Hold} is isolated from the opamp output at the end of that period. Each analog output bus is served by an analog output buffer with suitably high input impedance.

Combining the scaled references results shown in the schematic, the output is:

Equation 1

$$V_{Out} = (V_{REF} \pm AGND)\frac{C_1 C_4}{C_2 C_5} + (V_{REF} \pm AGND)\frac{C_3}{C_5} + AGND$$

When the global parameter RefMux is configured to (2 BandGap) ± BandGap in the Device Editor, AGND is 2.6V and the reference voltage is 1.3V. The corresponding output is:

Equation 2

$$2.6\,\mathrm{Volts} \pm 1.3\,\mathrm{Volts}\left(\frac{C_1}{2^5 \cdot 2^5} + \frac{C_3}{2^5}\right)$$

Equation 2 appears to be a result with 10 bits of magnitude; however, recall that C_1 is constrained to values obtained scaling the 3 least significant magnitude bits by a factor of four. Canceling the scale factor in C_1 and in its denominator, the result can be expressed as:

Equation 3

$$2.6\,\mathrm{Volts} \pm 1.3\,\mathrm{Volts}\left(\frac{C_1}{2^5 \cdot 2^5} + \frac{C_3}{2^5}\right), \qquad C_1 \in \{0, 1, 2, 3, 4, 5, 6, 7\}$$

Example

Equations 2 and 3 above show that the 9bit DAC input code is distributed across 2 values and the sign bit. The MSB of the input code is used as the sign bit. Bits 2 through 7 are use to specify the 5 bit value of C_3 and finally the three LSBs are used to specify C_1. For a input code value of +205 the sign is positive, the value of C3 is 25 and the value of C1 is 5. By applying these value to Equation 2, Equation 4 is obtained:

Equation 4

$$V_{Out} = 2.6\,\mathrm{Volts} + 1.3\,\mathrm{Volts}\left(\frac{5}{2^5 \cdot 2^5} + \frac{25}{2^5}\right) = 3.63\,\mathrm{Volts}$$

The value calculated is an ideal value and will most likely differ based on system noise and chip offsets.

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the table below, TA = 25°C and Vdd = 5V. Also, fclock = 125 kHz, external AGND 2.50V, external VRef 1.23V, REFPWR = HIGH, SCPOWER = ON, PSoC block power HIGH.

Table 1. 5.0V DAC9 DC and AC Electrical Characteristics, CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52Family of PSoC Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Resolution	--	9	Bits	
Linearity				
DNL	0.99	--	LSB	
INL	0.76	--	LSB	
Monotonic	Yes	--		
Gain Error				
Including Reference Gain Error	3.4		%FSR	
Excluding Reference Gain Error[1]	0.5		%FSR	
V_{OS}, Offset Voltage	±7.5	--	mV	
Output Noise	4.4	--	mV rms	0 to 300 kHz
f_{clock}, Analog Column Clock[2]				
Low Power	8 to 500	--	kHz	
Med Power	4 to 2000	--	kHz	
High Power	4 to 3200	--	kHz	
Operating Current[3]				
Low Power	300	--	µA	
Med Power	1130	--	µA	
High Power	4315	--	µA	

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the table below, TA = 25°C and Vdd = 3.3V. Also, fclock = 125 kHz, external AGND 1.50V, external VRef 0.8V, REFPWR = HIGH, SCPOWER = ON, PSoC block power HIGH.

Table 2. 3.3V DAC9 DC and AC Electrical Characteristics, CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52Family of PSoC Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Resolution	--	9	Bits	
Linearity				
DNL	0.99	--	LSB	
INL	075	--	LSB	
Monotonic	Yes	--		
Gain Error				
Including Reference Gain Error	2.6		%FSR	
Excluding Reference Gain Error[1]	0.3		%FSR	
V_{OS}, Offset Voltage	±7.5	--	mV	
Output Noise	3	--	mV rms	0 to 300 kHz
f_{clock}, Analog Column Clock[2]				
Low Power	8 to 500	--	kHz	
Med Power	4 to 2000	--	kHz	
High Power	4 to 3200	--	kHz	
Operating Current[3]				
Low Power	290	--	µA	
Med Power	1090	--	µA	
High Power	4175	--	µA	

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the table below, TA = 25°C and Vdd = 2.7V. Also, fclock = 125 kHz, external AGND 1.50V, external VRef 0.8V, REFPWR = HIGH, SCPOWER = ON, PSoC block power HIGH.

Table 3. 2.7V DAC9 DC and AC Electrical Characteristics, CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52Family of PSoC Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Resolution	--	9	Bits	
Linearity				
DNL	0.99	--	LSB	
INL	075	--	LSB	
Monotonic	Yes	--		
Gain Error				
Including Reference Gain Error	2.6		%FSR	
Excluding Reference Gain Error[1]	0.3		%FSR	
V_{OS}, Offset Voltage	±7.5	--	mV	
Output Noise	3	--	mV rms	0 to 300 kHz
f_{clock}, Analog Column Clock[2]				
Low Power	8 to 500	--	kHz	
Med Power	4 to 2000	--	kHz	
High Power	4 to 3200	--	kHz	
Operating Current[3]				
Low Power	290	--	µA	
Med Power	1090	--	µA	
High Power	4175	--	µA	

Electrical Characteristics Notes

1. Does not include reference block power, common to all analog blocks (see the PSoC Family data-sheet).
2. Upper end of range specified for 3dB increase in broadband noise. Lower end for droop < 1 LSB. The analog column clock selected by the DAC is 4 times faster than the phase clock rate that governs the update cycle. See the discussion of timing, below.
3. Reference Gain Error measured by comparing the external reference to $V_{RefHigh}$ and V_{RefLow} routed through the test mux and back out to a pin.

See the Placement section, in this user module, for a list of placements that meet these linearity and monotonicity specifications.

Placement

The DAC9 User Module maps onto two PSoC blocks designated LSB and MSB. The output of the LSB block is fed into the input of the MSB block therefore they are always placed adjacent to each other. In the CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52device family the MSB block maps only to “Type C" switched capacitor PSoC Blocks. This helps to minimize linearity errors because, these types of blocks permit the auto-zero process to cancel offset errors across the “BCap" capacitor (C_4 in the figure above) that couples the LSB and MSB blocks together.

An additional consideration, in selecting a placement location, is that the MSB and LSB clocks must be derived from the same source. This happens automatically, if they are placed in the same column in the analog array. If they are placed in two different columns, both column multiplexers must be set to the same source.

Parameters and Resources

To create a DAC9 instance, select the user module in the Device Editor, rename it if desired, and map it onto the device. Placement considerations include availability of an analog column output bus if the signal is to be driven off chip and interdependence with other user modules on the column clock resource(s). Once placed, the user module displays its parameters.

DataFormat

The DAC9 User Module API handles three different data formats: offset binary, 2’s complement, and sign-and-magnitude. The WriteBlind entry point of the API section (below), describes these conventions and the range of values associated with each.

AnalogBus

The DAC block broadcasts its output to adjacent analog PSoC blocks. Choosing one of the analog bus options connects the DAC output to the outside world through one of the analog output buffers. In certain columns, selecting the bus provides an additional local connection to the PSoC block at the top of the array. Switched capacitor PSoC blocks incorporate a sample and hold circuit that samples the DAC output in the last half of Φ_2. This isolates external outputs from the voltage swings that occur during auto-zero operation.

ClockPhase

This parameter determines the role of the phase clocks, Φ_1 and Φ_2, generated by the column clock divider discussed in the clock and timing sections that follow. When *Normal* is selected, the auto-zero cycle occurs during Φ_1 and the output of the DAC is valid in Φ_2. When ClockPhase is set to *Swapped*, these roles are reversed. This can be useful when the DAC is connected to another peripheral that samples its input on Φ_1.

Note Setting ClockPhase to Swapped disables the sample and hold function of the analog output bus. If the AnalogBus parameter is set to Enabled, the bus output mirrors the local PSoC block output, alternating between AGND (plus the offset voltage) during Φ_1 and the desired output during Φ_2.

Analog Column Clock

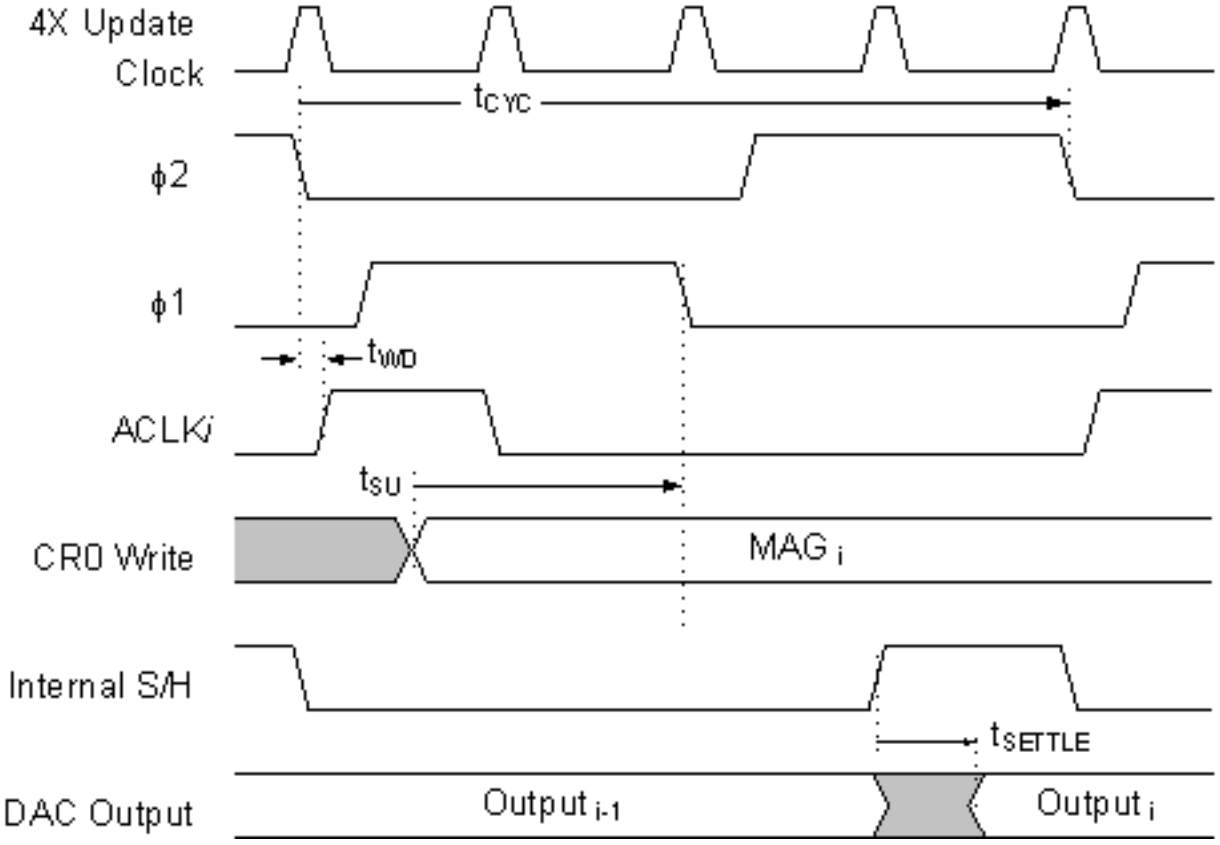
The DAC continuously updates its output whether or not it is commanded to “write" an new value by calling the appropriate functions WriteBlind and WriteStall API functions. The analog column clock multiplexors selects the source clock used to generate the phase clocks, Φ_1 and Φ_2 that control this update operation. The phase clock generator divides the column clock by four to produce Φ_1 and Φ_2,

so the column clock frequency is four times faster than the actual analog output update rate. Two levels of multiplexing provide choices for the column clock that include any of the digital blocks and the system clock dividers. The Electrical Characteristics section, above, specifies lower and upper limits for the column clock frequency.

Note To use DAC9 User Module at 5 V with a higher analog column clock frequency (fclock>1.5 MHz), the opamp bias should be set to High in the Global Resources settings to eliminate noise in the output

For positive output voltages (Vout > AGND) and normal phase, the reference voltage is stored on the A Cap during Φ_1 as shown in the simplified schematic, above. In order to fully charge the A Cap, any change to it's value must meet set-up time t_{SU} to the falling edge of Φ_1. This set-up time, illustrated in the figure below, is dependent on the reference power levels. Although the set-up time is not characterized, the hardware stall mechanism can be used to guarantee that it will be met. If the A Cap does not fully charge due to set-up time violations the output will be incorrect until the next phase clock cycle when the entire period of f1 when it will be corrected. A similar set-up time relative to the falling edge of f2 governs behavior for negative output voltages (Vout < AGND).

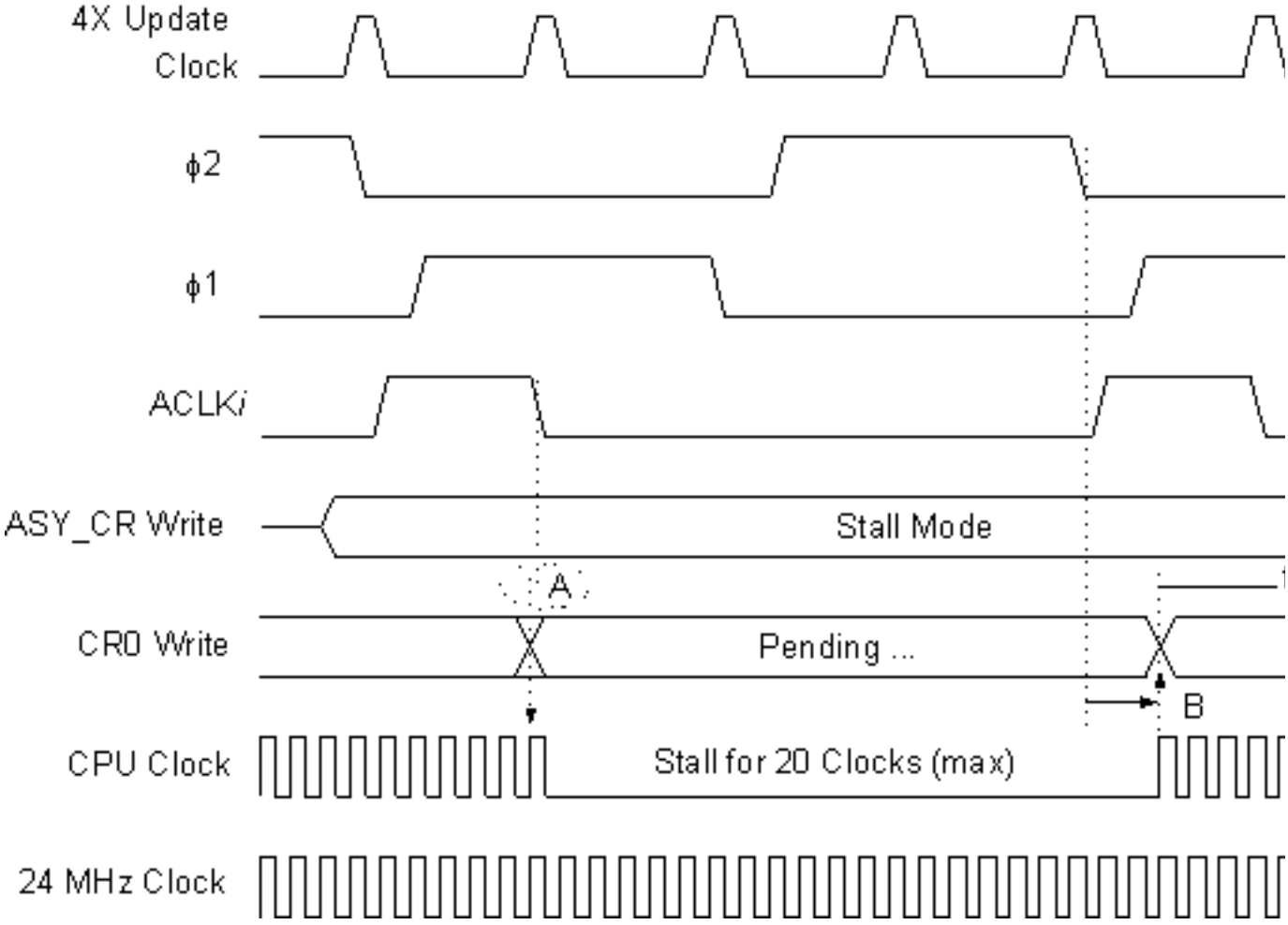
Figure 3. Update Timing for V_{OUT} > AGND



For a large class of applications, momentary (one update-cycle) deviations are acceptable. Other application may impose stricter requirements. Hardware synchronization may be employed to control the timing of the register write that changes the value of the A Cap. This is directly supported in the WriteStall API by entry point. When invoked, the underlying hardware recognizes the write to the

PSoC block register and freezes the CPU clock, holding off completion the write until the rising edge of Φ_1. The ASY_CR register controls.

Figure 4. Hardware Synchronization and CPU Clock Stall



During the CPU stall, all analog and digital PSoC blocks function normally. The MOV instruction that writes the DAC's CR0 register is simply suspended and, during this period, any interrupts become or remain pending. The number of CPU cycles lost during the stall equal, in time, the period of the can be calculated using the following relation.

Equation 5

$$\mathrm{CPU\,Cycles} \le \frac{F_{CPU}}{F_{\phi_1}} = \frac{4 \times F_{CPU}}{F_{ColClock}}$$

Clearly, to minimize the CPU cycles lost to the stall, the column clock should be run at the highest practical frequency. This can be much higher than necessary for actual changes to the output voltage as extra output cycles simply repeat the previous output cycle. A faster column clock also minimizes the latency from calling the output function to the time the output changes.

There are practical limits to the column clock frequency, however. Since the DAC must slew from AGND to the output voltage each phase-clock cycle, the column clock is limited to frequencies that permit the output to settle. When the sample and hold feature of the analog output bus is used, the

opamp output drives the bus during the sampling window in the second half of Φ_2. If the column clock is so fast that the op amp is still slewing to and settling on the output voltage, this becomes observable as noise on the output signal. In extreme cases, the output will slew only part way to the final output voltage before the sampling window closes. This may be observed as severe non-linear gain compression on the output most evident at the full-scale ends of the output range. Upper limits for the analog column clock that prevent this from occurring are provided in the Electrical Characteristics tables, above.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include" files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to perserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Entry points are provided to initialize the DAC9 User Module, write updated values, and disable the user module.

DAC9_Start

Description:

Performs all required initialization for this user module and sets the power level for the switched capacitor PSoC block.

C Prototype:

```
void  DAC9_Start(BYTE bPowerSetting)
```

Assembler:

```
mov   A, bPowerSetting
lcall  DAC9_Start
```

Parameters:

bPowerSetting: One byte that specifies the power level. Following reset and configuration, the PSoC block assigned to the DAC block is powered down. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value
DAC9_OFF	0
DAC9_LOWPOWER	1
DAC9_MEDPOWER	2
DAC9_FULLPOWER	3

Return Value:

None

Side Effects:

The DAC outputs will be driven. By default, the initial value is AGND. Call one of the Write routines prior to calling “Start" if some other output value is required at power on. The A and X registers may be altered by this function.

DAC9_Stop

Description:

Powers the user module Off.

C Prototype:

```
void  DAC9_Stop(void)
```

Assembly:

```
lcall  DAC9_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

Outputs will not be driven. The A and X registers may be altered by this function.

DAC9_SetPower

Description:

Sets the power level for the DAC switched capacitor PSoC block. May be used to turn the block Off and On. (This is the same as the Start function.)

C Prototype:

```
void  DAC9_SetPower(BYTE bPowerSetting)
```

Assembler:

```
mov   A, bPowerSetting
lcall  DAC9_SetPower
```

Parameters:

bPowerSetting: Identical to the PowerSetting parameter used for the Start entry point.

Return Values:

None

Side Effects:

The DAC outputs will be driven. By default, the initial value is AGND. Call one of the Write routines prior to calling “Start" if some other output value is required at power on. The A and X registers may be altered by this function.

DAC9_WriteBlind

Description:

Immediately updates the output voltage to the indicated value.

C Prototypes:

```
// For OffsetBinary:
void  DAC9_WriteBlind(INT iOutputValue)

// For TwosComplement:
void  DAC9_WriteBlind(INT iOutputValue)

// For TwoByteSignAndMagnitude in register format:
void  DAC9_WriteBlind2B(byte bLSB, byte bMSB)
```

Assembler:

```
; for OffsetBinary:
mov   X, >iOutputValue            ; upper byte
mov   A, <iOutputValue            ; lower byte
lcall  DAC9_WriteBlind
;or
mov   X, [piOutputValue]          ; upper byte
mov   A, [piOutputValue+1]        ; lower byte
lcall  DAC9_WriteBlind

; for TwosComplement:
mov   X, >iOutputValue            ; upper byte
mov   A, <iOutputValue            ; lower byte
lcall  DAC9_WriteBlind

; for TwoByteSignAndMagnitude format:
mov   X, bMSB                     ; MSB block CR0 register value
mov   A, bLSB                     ; LSB block CR0 register value
lcall  DAC9_WriteBlind
```

Parameters:

iOutputValue: Two bytes (each) that specify the output voltage. Allowed values lie in the range corresponding to the selected value of DataFormat as given below.

Data Format	Minimum	Maximum
OffsetBinary	0	510
TwosComplement	-255 (FF01h)	255 (00FFh)
TwoByteSignAndMagnitude	3F1Ch	1F3Ch

Offset-binary values are positive numbers with the lowest output voltage represented by 0 and the highest by 510. For 9-bit 2's complement, the upper byte is either 00h for positive values or FFh for negative values. In TwoByteSignAndMagnitude format, high byte takes the form $00smmmmm_2$ and the low byte $00tmmm00_2$, where 's' is the sign, 't' is the inverted sign, and 'm' represents magnitude bits. For positive numbers, s=0 and t=1.

Return Values:

None

Side Effects:

The output may glitch for reasons discussed in the Timing section in this user module. The A and X registers may be altered by this function.

Note: When you select the OFFSET_BINARY, input values that are out of range are automatically converted to two's complement data within the API. This means that an out of range value, above the maximum allowed offset binary value, is converted to a small positive output (near Agnd).

DAC9_WriteStall

Description:

Possibly stalls the microprocessor until the beginning of Φ_1, then updates the output voltage to the indicated value. Note that the API assumes that either interrupts are disabled or the maximum interrupt latency is less than ACLK*i* (see the Forced Synchronization with Fast Update Clock figure).

C Prototypes:

```
// For OffsetBinary:
void  DAC9_WriteStall(INT iOutputValue)

// For TwosComplement:
void  DAC9_WriteStall(INT iOutputValue)

// For TwoByteSignAndMagnitude in register format:
void  DAC9_WriteStall2B(byte bLSB, byte bMSB)
```

Assembly:

```
; for OffsetBinary:
mov   X, >iOutputValue                ; upper byte
mov   A, <iOutputValue                ; lower byte
lcall  DAC9_WriteStall
;or:
mov   X, [piOutputValue]              ; upper byte
mov   A, [piOutputValue+1]            ; lower byte
lcall  DAC9_WriteStall

; for TwosComplement:
```

```
mov   X, >iOutputValue                ; upper byte
mov   A, <iOutputValue                ; lower byte
lcall  DAC9_WriteStall

; for TwoByteSignAndMagnitude format:
mov   X, bMSB                          ; MSB block CR0 register value
mov   A, bLSB                          ; LSB block CR0 register value
lcall  DAC9_WriteStall
```

Parameters:

wOutputValue or iOutputValue: Identical in format and value range to the parameters described for the WriteBlind entry point. bMSB and bLSB: Identical in format and value range to the parameters described for the WriteBlind entry point.

Return Values:

None

Side Effects:

If ACLKi is inactive (where 'i' is the column into which the analog PSoC block is mapped), the microprocessor's CPU clock is disabled until Φ_2 goes inactive, possibly for three-quarters of an update cycle (plus two CPU clocks). Note that no interrupts are recognized during the stall interval. The A and X registers may be altered by this function.

Sample Firmware Source Code

The sample code creates a periodic, descending sawtooth wave.

```
;;-------------------------------------------------------------------
;; Sample Assembly Code for the DAC9
;; Generate a falling sawtooth wave
;;-------------------------------------------------------------------

include "m8c.inc"
include "DAC9.inc"

          area bss (RAM)
iVal::    blk  2                 ; RAM for loop iteration variable
iMAXVAL:: equ  511               ; Top of ramp plus 1
          area text (ROM, REL)

_main::                          ; (contains infinite loop; never returns)
   mov  A, DAC9_LOWPOWER         ; specify DAC's amplifier power
   call DAC9_Start               ; and turn it on.

Init:
                                 ; Start ramp from the top
   mov  [iVal],   >iMAXVAL       ; upper byte
   mov  [iVal+1], <iMAXVAL       ; lower byte

RampDown:
   dec  [iVal+1]                 ; decrement lower byte
   jnc  skip_dec_wVal_upper      ; if lower goes from 00h to FFh, dec upper
   dec  [iVal]                   ; decrement upper byte
   jc   Init                     ; if upper goes from 00h to FFh, start over
```

```
skip_dec_wVal_upper:
   mov  X, [iVal]                ; upper byte
   mov  A, [iVal+1]              ; lower byte
   call DAC9_WriteStall
   jmp  RampDown                 ; next step
```

The same code in C is:

```
//-------------------------------------------------------------------
// Sample C Code for the DAC9
// Generate a rising sawtooth wave.
//-------------------------------------------------------------------

#include "m8c.h"
#include "DAC9.h"

#if DAC9_OFFSETBINARY
    const INT  iMAXVAL=510;         // top of ramp
    const INT  iMINVAL=0;           // bottom of ramp
#else
    #if DAC9_TWOSCOMPLEMENT
        const INT  iMAXVAL=255;     // top of ramp
        const INT  iMINVAL=-255;    // bottom of ramp
    #else
        // DAC9_SIGNANDMAGNITUDE
    #endif
#endif


void main(void)
{
   INT  iVal;                   // RAM for loop iteration variable

DAC9_Start(DAC9_FULLPOWER);   // power up the DAC

   if(DAC9_OFFSETBINARY)

   {
      while(1)
      {
         for(iVal=iMINVAL; iVal <= iMAXVAL; iVal++)
            DAC9_WriteStall(iVal);                    // update DAC
      }
   }
   else if(DAC9_TWOSCOMPLEMENT)
{
      while(1)
      {
         for(iVal=iMINVAL; iVal <= iMAXVAL; iVal++)
         {
       DAC9_WriteStall(iVal);                    // update DAC
         }
      }
   }
   else
```

```
    {
        DAC9_Stop();                                    // power off DAC
    }
}
```

Configuration Registers

The API provides a complete interface to the DAC. Writing directly to the configuration registers affords an alternative means of updating the output. Either way, there are timing considerations which must be understood to prevent output glitches. The following registers are used for the DAC9 switched capacitor DAC block.

Table 4. Block LSB: Register CR0

Bit	7	6	5	4	3	2	1	0
Value	1	0	Sign	Magnitude			0	0

Sign uses a '1' for positive values (AGND up to RefHi) and a '0' for negative values (RefLow up to AGND). The default is '1'. Note that this is opposite the sense used in the MSB Block. Use one of the Write functions in the API to change Sign. Magnitude's default is '0'. Use one of the Write functions in the API to change Magnitude.

Table 5. Block LSB: Register CR1

Switched Cap Type A								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	0	1	0	0	0	0	0	0
Switched Cap Type B								
Bit	7	6	5	4	3	2	1	0
Value	1	0	0	0	0	0	0	0

Table 6. Block LSB: Register CR2

Bit	7	6	5	4	3	2	1	0
Value	Analog Bus	0	1	0	0	0	0	0

AnalogBus is enabled or disabled at configuration time in the Device Editor.

Table 7. Block LSB: Register CR3

Switched Cap Type A								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	0	0	1	1	0	0	Power	
Switched Cap Type B								
Bit	7	6	5	4	3	2	1	0
Value	0	0	1	1	1	0	Power	

Power: The default is Off. Use the Start call in the API to set this value.

Table 8. Block MSB: Register CR0

Bit	7	6	5	4	3	2	1	0
Value	1	0	Sign	Magnitude				

Sign is set by the API Write routines. Magnitude is changed by using one of the Write functions in the API.

Table 9. Block MSB: Register CR1

Bit	7	6	5	4	3	2	1	0
Value	0	1	0	0	0	0	0	1

Table 10. Block MSB: Register CR2

Bit	7	6	5	4	3	2	1	0
Value	Analog Bus	0	1	0	0	0	0	0

AnalogBus is enabled or disabled at configuration time in the Device Editor.

Table 11. Block MSB: Register CR3

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	1	BMux		Power	

BMux is configured to select the connection from the LSB PSoC block. Power: 0 = Off, 1 = Low, 2 = Medium, 3 = Full. The default is Off. Use the Start call in the API to set this value.

Version History

Version	Originator	Description
2.2	DHA	Added Version History

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2001-2019 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.